Route Feasibility Testing and Forward Time Slack for the Synchronized Pickup and Delivery Problem

Timo Gschwind[a]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

Abstract

The Synchronized Pickup and Delivery Problem (SPDP) consists of finding a set of minimum-cost routes servicing user-specified transportation requests from pickup to delivery locations subject to pairing and precedence, capacity, time-window, and minimum and maximum time-lag constraints. The temporal constraints of the SPDP impose a complex scheduling problem for the service times at the customer locations which makes the efficient feasibility checking of routes intricate. We present different route feasibility tests for the SPDP and compare their practical runtime on a huge number of randomly generated routes. Furthermore, we generalize to the SPDP the concept of forward time slack, which has proven a versatile tool for feasibility testing of customer or request insertions into a given (feasible) route for many VRP variants.

Key words: Vehicle routing, Temporal synchronization, Feasibility testing, Forward time slack

1. Introduction

The Synchronized Pickup and Delivery Problem (SPDP, Gschwind, 2015) is the prototypical Vehicle Routing Problem (VRP) with *temporal intra-route synchronization* constraints. It seeks to find a set of minimum-cost routes servicing n user-specified transportation requests from origin (or pickup) to destination (or delivery) points subject to pairing and precedence, capacity, and time-window constraints. Moreover, the service times at the pickup and delivery locations of the customer requests are synchronized in the following way: After completing the service at a pickup point, the corresponding delivery has to be performed within prespecified *minimum* and *maximum time lags* (or *Ride Times*, RTs). From a modeling point of view, the SPDP generalizes the Dial-A-Ride Problem (DARP, see Cordeau and Laporte, 2007, for a survey) in which no minimum RTs are present.

Applications of the SPDP arise, e.g., in home health care where patients need to be monitored repeatedly within given time intervals (Eveborn *et al.*, 2006; Rasmussen *et al.*, 2012) and, for personnel consistency reasons, it may be necessary that these visits are always performed by the same nurse (Kovacs *et al.*, 2014). Similar problems can be found in the planning of security guards where locations have to be inspected several times a day/night (Bredström and Rönnqvist, 2008). In the airline industry, minimum and maximum RTs are relevant, e.g., in fleet assignment models or crew scheduling problems. In the former, they constrain the allowed time between consecutive flights serving the same origin-destination pair (*spacing constraints,* Bélanger *et al.*, 2006). In the latter, they correspond to minimum and maximum idle times between two sequential flights of the same crew (Barnhart *et al.*, 2003).

Because of the complex temporal constraints of the SPDP, deciding whether or not a given route is feasible is a non-trivial task. The efficient feasibility testing of routes, however, is a crucial part in many exact and heuristic algorithms for VRPs. Sophisticated feasibility tests for a DARP with an additional constraint on the maximum waiting time at the customer nodes have been proposed by Tang *et al.* (2010)

Email address: gschwind@uni-mainz.de (Timo Gschwind)

and Firat and Woeginger (2011). The time complexity of these approaches is $\mathcal{O}(n^2)$ and $\mathcal{O}(n)$, respectively. In the SPDP, the additional presence of minimum RT constraints further complicates the route feasibility problem.

Another crucial aspect for some solution approaches to VRP variants is the ability to quickly evaluate the feasibility of insertions of single nodes or requests into a given (feasible) route. The concept of *Forward Time Slack* (FTS) originally introduced by Savelsbergh (1992) for the VRP with Time Windows (VRPTW) can be a useful tool for this kind of evaluation. Generalized versions of the FTS have been used to asses the feasibility of insertions in heuristic algorithms, e.g., for the Technician Routing and Scheduling Problem (Pillac *et al.*, 2012) and the Pickup and Delivery Problem with Transfers (PDPT) (Masson *et al.*, 2013). In a companion paper (Gschwind, 2015), the FTS principle is used within a dynamic-programming labeling algorithm for the solution of the column-generation subproblem of the SPDP.

The contribution of this paper is twofold. First, we derive different route feasibility checks for the SPDP by adapting the approaches for the DARP presented in Firat and Woeginger (2011) and Tang *et al.* (2010) and conduct a computational study over a large number of randomly generated routes to compare the practical runtime of the proposed procedures. Our results indicate that the adapted versions of the algorithm of Tang *et al.* (2010) clearly outperform the other approaches. Second, we generalize the concept of FTS to the SPDP and demonstrate why the definition of the FTS is not unique for problems with maximum RT constraints.

The remainder of the paper is structured as follows: Section 2 defines the SPDP. Section 3 derives feasibility tests for individual routes and presents results comparing their practical runtime. The adaptation of the FTS concept to the SPDP is described in Section 4. Short conclusions are drawn in Section 5.

2. Problem Definition

Note again, that this paper is not about solving the SPDP but on route feasibility testing and FTS. For the sake of completeness, however, we still provide a definition of the problem as a whole in the following. The SPDP is defined on a complete digraph $G = (V, A)$ with node set V and arc set A. The node set V comprises the origin and destination depots 0 and $2n+1$, the set of pickup nodes $P = \{1, ..., n\}$, and the set of delivery nodes $D = \{n+1, ..., 2n\}$. For each node $i \in V$ a time window $[a_i, b_i]$, a service duration s_i, and a demand q_i with $q_i = -q_{i+n}$ are given. We assume w.l.o.g. that the beginning of the planning horizon is at time zero, i.e., $a_0 = 0$. A travel time t_{ij} and a routing cost c_{ij} are associated with each arc $(i, j) \in A$. Both travel times and routing costs are assumed to satisfy the triangle inequality.

Each transportation request $i = 1, ..., n$ consists of transporting a specific good from the pickup node $i \in P$ to the delivery node $i+n \in D$. A minimum RT $\underline{L}_i$ and a maximum RT $\overline{L}_i$ are associated with each transportation request i.

A fleet K of homogeneous vehicles each with a capacity of Q is located at the origin depot 0 to serve the transportation requests. The task of the SPDP is to find a set of $|K|$ routes starting and ending at the depot nodes 0 and $2n+1$ such that each transportation request is performed exactly once. Thereby, vehicle capacities have to be respected and the service at each node has to be started within its time window. If a vehicle arrives prior to a_i at node i, it has to wait until the time window opens. Moreover, waiting, i.e., voluntarily delaying the start of service, is allowed at any node at any time. For each transportation request i, the pickup and delivery nodes have to be served on the same route and the pickup has to be serviced before. Furthermore, the service at the delivery node $i+n$ has to be started at least $\underline{L}_i$ and at most $\overline{L}_i$ units of time after the service at the pickup has been completed.

3. Route Feasibility Testing

Consider a given route $R = (v_1, ..., v_{2r+2})$ with $v_1 = 0$ and $v_{2r+2} = 2n+1$. We use the notation $v_i \in R$ to indicate that a node v_i is part of route R. Furthermore, given a pickup (delivery) node $v_i \in P$ ($v_i \in D$), we indicate by v_{i^-} and i^- (v_{i^+} and i^+) the corresponding delivery (pickup) node and its associated notation, while the corresponding request is referred to by using the visit index i of node v_i.

Testing the feasibility of a route R means checking its consistency with all constraints of the SPDP that relate to individual routes. The verification of pairing and precedence as well as the capacity constraint is independent of each other and of the temporal constraints. For these types of constraints the verification in linear time is straightforward. We assume in the following that they are respected. Checking whether or not the temporal constraints are satisfied is intricate. Indeed, one has to verify if there exists a time schedule $\mathcal{T}_R = (\tau_1, ..., \tau_{2r+2})$ satisfying

$$a_{v_i} \leq \tau_i \leq b_{v_i} \quad \forall v_i \in R, \tag{1}$$

$$\tau_i + s_{v_i} + t_{v_i v_{i+1}} \leq \tau_{i+1} \quad \forall v_i \in R, i \neq 2r+2, \tag{2}$$

$$\tau_i + s_{v_i} + \overline{L}_{v_i} \geq \tau_{i^-} \quad \forall v_i \in R \cap P, \tag{3}$$

$$\tau_i + s_{v_i} + \underline{L}_{v_i} \leq \tau_{i^-} \quad \forall v_i \in R \cap P, \tag{4}$$

where τ_i denotes the start of service at node v_i. This is called the *scheduling problem* of the SPDP. Constraints (1) are time-window constraints. Consistency of the service times along the route is ensured by constraints (2), while (3) and (4) are maximum and minimum RT constraints, respectively. For ease of notation, we assume service durations of zero and omit them in all formulae in the following. The extension of all concepts to non-zero service durations is straightforward.

In Tang *et al.* (2010) and Firat and Woeginger (2011), feasibility tests for a similar scheduling problem, i.e., with constraints (1)–(3) and an additional constraint on the waiting time at each node $v_i \in R$, were presented. In the following, we sketch both algorithms and extend them to solve the scheduling problem of the SPDP. Also, we highlight the increased complexity coming from the additional presence of the minimum RT constraints that leads to increased worst-case running times for some of the adapted procedures.

3.1. Adapted Feasibility Test of Firat and Woeginger

The basic idea of Firat and Woeginger (2011) is to rewrite the considered scheduling problem as a system of difference constraints. It is well-known (see, e.g., Cormen *et al.*, 2001, Section 24.4) that such a system has a solution if and only if an associated digraph (called *constraint graph*) has no negative-weight cycle. Moreover, they formulate the difference-constraint system over an appropriate set of variables allowing the cycle-detection test to be performed in linear time by a transformation of the constraint graph into a specific interval graph.

Note that Firat and Woeginger start from a slightly more complex formulation of the scheduling problem using variables for the arrival times, the departure times, and the actual service times at the nodes. The service time at a given node is then required to lie between arrival and departure times at that node. It is easy to verify, however, that the condensed formulation (1)–(3) using only variables for the service times is equivalent to their formulation. Even more, the slightly different formulation of the scheduling problem leads to a constraint graph with the same general structure that was exploited by Firat and Woeginger (2011). This is also true when adding minimum RT constraints to both formulations.

Before we sketch their approach, some additional notation is necessary. For each node $v_i \in R$, the constant T_i denotes the sum of the travel times along the route R up to node v_i. The total waiting time up to node $v_i \in R$ is given by w_i.

Original Algorithm for the DARP. The first step of Firat and Woeginger (2011) is to rewrite constraints (1)–(3) of the considered scheduling problem in terms of T_i and w_i yielding

$$a_{v_i} \leq T_i + w_i \leq b_{v_i} \quad \forall v_i \in R, \tag{5}$$

$$w_i - w_{i+1} \leq 0 \quad \forall v_i \in R, i \neq 2r+2, \tag{6}$$

$$w_{i^-} - w_i \leq T_i + \overline{L}_{v_i} - T_{i^-} \quad \forall v_i \in R \cap P. \tag{7}$$

Next, two additional dummy variables w_0 and w_{2r+3} are introduced representing values $w_0 = 0$ and $w_{2r+3} = b_{2n+1} - a_0$, i.e., they constitute lower and upper bounds for all $w_i, i = 1, ..., 2r+2$. Using w_0, w_{2r+3}, and

defining $M = b_{2n+1} - a_0$, constraints (5) can be written as the difference constraints

$$w_{2r+3} - w_i \leq M + T_i - a_{v_i} \quad \forall v_i \in R, \tag{8}$$
$$w_i - w_0 \leq b_{v_i} - T_i \quad \forall v_i \in R. \tag{9}$$

The final set of difference constraints is (6)–(9) together with

$$w_{2r+3} - w_0 \leq M, \tag{10}$$
$$w_0 - w_{2r+3} \leq -M, \tag{11}$$

to enforce $w_{2r+3} - w_0 = M$. The constraint graph associated with this system has a node for each $w_i, i = 0, ..., 2r+3$ and an arc with weight d_{ij} from node w_j to node w_i for each constraint $w_i - w_j \leq d_{ij}$ of the system (6)–(11). It has $\mathcal{O}(r)$ nodes and $\mathcal{O}(r)$ arcs. Thus, negative-weight cycles can be detected in $\mathcal{O}(r^2)$, e.g., by using the Bellman-Ford algorithm (BF).

The key observation of Firat and Woeginger (2011) to obtain a linear-time feasibility check is the following: They call an arc from w_j to w_i a forward arc if $j < i$, otherwise it is a backward arc. All arcs in the constraint graph are either forward arcs with non-negative weights (otherwise the scheduling problem would be trivially infeasible) or backward arcs with weight zero. The only exception is the arc corresponding to constraint (11) which is a backward arc with negative weight. Using this structure, they are able to transform the graph into a specific interval graph for which the cycle-detection test can be done in linear time.

Adapted Algorithm for the SPDP. The adaptation of the approach to the SPDP requires to perform the same transformation described above also to constraints (4) yielding:

$$w_i - w_{i^-} \leq T_{i^-} - T_i - \underline{L}_{v_i} \quad \forall v_i \in R \cap P. \tag{12}$$

The additional difference constraints (12) correspond to additional arcs in the constraint graph. Still, the total number of arcs is $\mathcal{O}(r)$ and the application of BF gives a $\mathcal{O}(r^2)$ feasibility test.

To show that feasibility testing in linear time is not possible for the SPDP (with the technique of Firat and Woeginger, 2011), we have to analyze the new arcs: They are backward arcs and have non-negative weight if $T_{i^-} - T_i \geq \underline{L}_{v_i}$, i.e., if the minimum RT of request i is trivially satisfied on the given route because the travel time from v_i to v_{i^-} is already larger than $\underline{L}_{v_i}$. Otherwise, the arc weight is negative. As a result, the specific structure of the constraint graph that was exploited to obtain a linear time cycle-detection test is lost.

Note that a direct reformulation, i.e., using the original variables, of the scheduling problem (1)–(4) as a system of difference constraints also leads to a constraint graph with the same number of nodes and arcs as in the approach of Firat and Woeginger (2011). Thus in the additional presence of minimum RTs, both reformulations lead to equivalent feasibility tests. Moreover, the direct reformulation as a system of difference constraints is essentially identical to the modeling of the feasibility problem as a simple temporal problem as proposed by Masson *et al.* (2014) for a DARP with transfer possibilities between routes.

Summing up, we have the following result.

Remark 1. *The different transformations into cycle-detection problems and their direct solution with appropriate shortest-path algorithms yield $\mathcal{O}(r^2)$ feasibility tests for the scheduling problem (1)–(4) of the SPDP.* □

However, these algorithms require building a constraint graph for each route to be tested which might negatively affect the practical runtime. In Section 3.3, we evaluate the practical runtime of two cycle-detection algorithms for feasibility testing of SPDP-routes. The first algorithm is based on a straightforward implementation of basic BF that detects negative cycles by solving a shortest-path problem in the constraint graph using a labeling strategy (see, e.g., Cherkassky *et al.*, 2009, for details on the general labeling method). The second algorithm uses an implementation of BFCT (Tarjan, 1981) which is a variant of BF and is one of

Algorithm 1: Algorithm of Tang *et al.* (2010) for the DARP

Result: **true** if feasible schedule exists, **false** otherwise

// Pass 1 (forward)

1 $\tau_1 := a_{v_1}$

2 **for** $i = 2, ..., 2r+2$ **do**

3 $\quad \tau_i := \max\left\{\tau_{i-1} + t_{v_{i-1}v_i}, a_{v_i}\right\}$

4 $\quad$ **if** $\tau_i > b_{v_i}$ **then return** false

// Pass 2 (backward)

5

6

7

8 **for** $i = 2r+1, ..., 1$ **do**

9 $\quad$ **if** $v_i \in P$ **then**

10 $\quad\quad \Delta_i := \tau_{i^-} - \tau_i - \overline{L}_{v_i}$

11 $\quad\quad$ **if** $\Delta_i > 0$ **then**

12 $\quad\quad\quad \tau_i := \tau_i + \Delta_i$

13

14 $\quad\quad\quad$ **if** $\tau_i > b_{v_i}$ **then return** false

15

16

17 $\quad\quad\quad$ **for** $j = i+1, ..., 2r+2$ **do**

18 $\quad\quad\quad\quad \tau_j := \max\left\{\tau_{j-1} + t_{v_{j-1}v_j}, a_{v_j}\right\}$

19 $\quad\quad\quad\quad$ **if** $\tau_j > b_{v_j}$ **then return** false

20 $\quad\quad$ **if** $\tau_{i^-} - \tau_i - \overline{L}_{v_i} > 0$ **then return** false

21 **return** true

Algorithm 2: Adapted algorithm of Tang *et al.* (2010) for the SPDP without inner forward loop

Result: **true** if feasible schedule exists, **false** otherwise

// Pass 1 (forward)

1 $\tau_1 := a_{v_1}$

2 **for** $i = 2, ..., 2r+2$ **do**

3 $$\tau_i := \begin{cases} \max\left\{\tau_{i-1} + t_{v_{i-1}v_i}, a_{v_i}, \tau_{i^+} + \underline{L}_{v_{i^+}}\right\} & \text{if } v_i \in D, \\ \max\left\{\tau_{i-1} + t_{v_{i-1}v_i}, a_{v_i}\right\} & \text{otherwise.} \end{cases}$$

4 $\quad$ **if** $\tau_i > b_{v_i}$ **then return** false

// Pass 2 (backward)

5 $i^* := 2r+1,\ cnt := 0$

6 **while** $i^* > 0$ **do**

7 $\quad i^* := -1,\ cnt := cnt + 1$

8 $\quad$ **for** $i = 2r+1, ..., 1$ **do**

9 $\quad\quad$ **if** $v_i \in P$ **then**

10 $\quad\quad\quad \Delta_i := \tau_{i^-} - \tau_i - \overline{L}_{v_i}$

11 $\quad\quad\quad$ **if** $\Delta_i > 0$ **then**

12 $\quad\quad\quad\quad \tau_i := \tau_i + \Delta_i$

13 $\quad\quad\quad\quad i^* := i$

14 $\quad\quad\quad\quad$ **if** $\tau_i > b_{v_i}$ **then return** false

15 $\quad$ **if** $i^* > 0$ **then**

16 $\quad\quad$ **if** $cnt > 2r$ **then return** false

17 $\quad\quad$ **for** $j = i^* + 1, ..., 2r+2$ **do**

18 $$\quad\quad\quad \tau_j := \begin{cases} \max\left\{\tau_{j-1} + t_{v_{j-1}v_j}, a_{v_j}, \tau_{j^+} + \underline{L}_{v_{j^+}}\right\} & \text{if } v_j \in D, \\ \max\left\{\tau_{j-1} + t_{v_{j-1}v_j}, a_{v_j}\right\} & \text{otherwise.} \end{cases}$$

19 $\quad\quad\quad$ **if** $\tau_j > b_{v_j}$ **then return** false

20

21 **return** true

the best performing cycle-detection algorithms (Cherkassky *et al.*, 2009). In BFCT, it is checked for negative cycles whenever a node is relabeled. This can be done with little additional effort and does not increase the $\mathcal{O}(r^2)$ total running time by using auxiliary data that can be maintained in amortized constant time.

3.2. Adapted Feasibility Test of Tang et al.

Tang *et al.* (2010) proposed a different feasibility test for the same scheduling problem as considered by Firat and Woeginger (2011), but with weaker, quadratic worst-case runtime. However, their algorithm seems more intuitive as it gradually constructs the schedule $\mathcal{T}_R$ directly on the original route/network. This also means that some information that is needed might already be available within an exact or heuristic approach in which the feasibility test is employed. Therefore, their algorithm might be sufficiently fast (or even faster than the algorithm of Firat and Woeginger, 2011) in practice, especially if the given routes are not too long (see, e.g., Gschwind and Irnich, 2015)).

Again, we first sketch the original algorithm before we present our adaptations to the SPDP.

Original Algorithm for the DARP. The algorithm of Tang *et al.* (2010) tries to construct a feasible schedule $\mathcal{T}_R$ satisfying (1)–(3) by traversing the route twice: once forward and once backward. If no feasible schedule can be found, the route is infeasible. Note that the third traversal of the original algorithm is redundant.

The whole procedure is described in Algorithm 1. The forward pass (Steps 1–4) builds a schedule of service times τ_i that satisfy constraints (1) and (2). Thereby, all times are scheduled as early as possible.

The backward pass (Steps 8–20) checks for consistency with the maximum RT constraints (3) adjusting some values τ_i if necessary: At each pickup node $v_i \in P \cap R$ it is checked if the current schedule satisfies the maximum RT of request i (Steps 9–11). If not, the algorithm tries to shift waiting time that occurs between pickup and delivery of request i before the pickup node v_i in order to decrease the ride time of request i. Thereto, the service at node v_i is delayed by as much as necessary to meet the maximum RT (Step 12). This requires shifting the service times τ_j of all succeeding nodes $v_j, j = i+1, ..., 2r+2$ forward in time (Steps 17–19) and is done in the same fashion as in the first pass, i.e., accounting for constraints (1) and (2). If there is not enough waiting time between pickup and delivery of request i, the maximum RT of i cannot be satisfied and the route is infeasible (Step 20). Note that this adjustment of the service times can never increase the RT of a request that is picked up later than v_i meaning that the maximum RT constraint of such a request never gets violated. Thus, after traversing a pickup node v_i in the backward pass we have a schedule that respects constraints (1) and (2), and the maximum RT constraints (3) of those requests for which the pickup node is not visited before v_i in the route. Moreover, all service times are still scheduled as early as possible with respect to the constraints that are satisfied at that point of the algorithm. Consequently, whenever a rescheduling results in a service time $\tau_j > b_{v_j}$ no feasible schedule exists and the route is infeasible.

Both forward and backward pass traverse the route once. Because of the inner forward loop (Steps 17–19), the backward pass has a quadratic worst-case running time and, hence, the overall algorithm also has time complexity $\mathcal{O}(r^2)$.

Adapted Algorithm for the SPDP. Solving the scheduling problem of the SPDP with the technique of Tang *et al.* (2010) requires the integration of the minimum RT constraints into the scheduling process. The adapted procedure is detailed in Algorithm 2 and is presented side-by-side with the original algorithm for the DARP to highlight the necessary modifications: In the forward pass, Step 3 changes to

$$\tau_i := \begin{cases} \max\left\{\tau_{i-1} + t_{v_i v_{i-1}}, a_{v_i}, \tau_{i^+} + \underline{L}_{v_{i^+}}\right\} & \text{if } v_i \in D, \\ \max\left\{\tau_{i-1} + t_{v_i v_{i-1}}, a_{v_i}\right\} & \text{otherwise.} \end{cases}$$

The resulting service times after the first pass satisfy constraints (1), (2), and (4) and they are scheduled in an earliest-possible fashion.

In the backward pass, whenever a shifting of service times is necessary because of some violated maximum RT (Steps 11–20), we need to change Step 18 in the same way as Step 3 in order to maintain feasibility with respect to constraints (1), (2), and (4). In contrast to the original algorithm, however, the shifting of waiting times can increase the RT of requests that are picked up later in the route. Decisive is that we might be forced to re-introduce waiting time somewhere in the route due to the minimum RT constraints of other requests. As a result, the property that after traversing a pickup node v_i in the backward pass all maximum RTs of requests which are not picked before v_i are respected, is lost.

Consider the example given in Table 1. The travel times between all nodes are assumed to be 10. The maximum RTs of requests i and m are 30. The minimum RT of request j is 52. The time window of each node is specified in Table 1. All other constraints are assumed to be never binding.

Table 1 gives the service times at each node at different stages of the algorithm. Waiting times at nodes are in brackets. Pairs of service times that do not satisfy the RT constraints are highlighted in italics. After the first pass of the algorithm, the maximum RT constraint of requests i and m are violated by 1 unit of time. Therefore, the second pass successively delays the service times τ_m at m and τ_i at i by 1 unit trying to shift waiting time that occurs between m and m^- before m and between i and i^- before i, respectively. However, the removed waiting time at node k decreases the RT of request j so that it has to be re-introduced before starting the service at node j^-. Otherwise, the minimum RT constraint of j would be violated. As a consequence, the RT of request m increases again and the schedule after the backward pass is infeasible which, however, does not imply that the route is infeasible.

A straightforward way to fix this defect of the algorithm is to loop over the second pass as long as an adjustment of the service times was necessary because of a violated maximum RT (Steps 6–20). For the

	Node	0	i	j	k	i^-	m	k^-	j^-	m^-	$2n+1$
	$[a.,b.]$	$[0,100]$	$[10,20]$	$[20,30]$	$[31,41]$	$[41,51]$	$[51,61]$	$[62,72]$	$[72,82]$	$[82,92]$	$[0,100]$
	$\underline{L}./\overline{L}.$	—	$0/30$	$52/100$	$0/100$	—	$0/30$	—	—	—	—
After pass 1	$\tau.$	0	*10*	20	(1) 31	*41*	*51*	(1) 62	72	*82*	92
1st pass 2, at node m	$\tau.$	0	*10*	20	(1) 31	*41*	(1) 52	62	72	82	92
1st pass 2, at node i	$\tau.$	0	(1) 11	21	31	41	(1) *52*	62	(1) 73	*83*	93
After 2nd pass 2	$\tau.$	0	(1) 11	21	31	41	(2) 53	63	73	83	93

Table 1: Example showing the necessity to loop over second pass in Algorithm 2

example above, one additional backward pass identifies a feasible schedule (see Table 1). This modification has two major effects. First, it is no longer mandatory to perform the inner forward loop (Steps 17–19) of the backward pass directly after every shifting of a service time due to a violated maximum RT. Alternatively, the update of the service times at all nodes v_j with $j > i^*$ can be done only once, after the backward traversal in each iteration of the second pass. Thereby, v_{i^*} is the pickup node with the smallest index i^* for which a delay of the service time τ_{i^*} was necessary in Step 12. This variant of the algorithm is presented in Algorithm 2. It generally requires to iterate over the second pass more often than with the inner forward loop. The complexity of Pass 2 itself, however, decreases to linear. In Section 3.3, we compare the practical performance of the versions with (*Algo2-w*) and without (*Algo2-w/o*) the inner forward loop performed after each adjustment in Pass 2.

The second and more severe consequence of having to loop over Pass 2 is that it leads to a significant deterioration of the worst-case complexity if the number of iterations of Pass 2 cannot be bounded. Consider the example as given in Table 2. The time windows of the nodes and the ride-time constraints of the requests are specified in the table. All other data is equivalent to the previous example. Apparently, the three requests i, j, and m are nested in a way that Algorithm 2 cyclically shifts waiting times that occur in between pickup and delivery of the requests i and m. In the example, this continues until one of the time windows is violated. Generally, this can lead to an unbounded number of iterations and, hence, a superlinear runtime of the algorithm.

In the following, we show that it is, however, possible to derive an upper bound on the number of iterations of Pass 2 that are necessary to decide on the (in)feasibility of a route. Recall that a general infeasibility certificate for the scheduling problem is the presence of a negative-weight cycle (cf. Cormen *et al.*, 2001, Section 24.4). In Table 2, e.g., the cyclic shifting of waiting times is caused by the constraints linking the nodes (i, j, j^-, m^-, m, i^-). It is easy to verify that these constraints form a negative-weight cycle (the weight of the cycle is -1). To see that a negative-weight cycle in the scheduling problem always leads to a cyclic shifting of waiting times in Algorithm 2, note that Algorithm 2 can be interpreted as a shortest path labeling algorithm (like, e.g., BF or BFCT) on the constraint graph associated with the difference-constraint system resulting from a direct reformulation of the scheduling problem (1)–(4): Whenever the service time τ_i of a node v_i is adjusted for a certain constraint of (1)–(4) in any step of Algorithm 2, this is equivalent to a relabeling of the corresponding node in the constraint graph due to the 'relax' operation in BF of the arc associated with this constraint. Thus, Algorithm 2 essentially is an alternative approach to implicitly solve the shortest path/cycle-detection problem in the constraint graph. Thereby, it uses knowledge about the problem structure to only check those constraints (i.e., arcs in the constraint graph) that are relevant for other service times after a specific service time (i.e., node label in the constraint graph) has been adjusted. Moreover, several relevant constraints are checked (i.e., relax operations are performed) simultaneously in a single step, e.g., in Step 3 of Algorithm 2: Time window of node v_i, travel time from node v_{i-1} to v_i, and minimum RT of request i. Also, the algorithm immediately stops once a time window is violated without the need to explicitly identify a cycle that includes the corresponding time-window constraint.

We now comment on how to detect the presence of a negative-weight cycle with Algorithm 2. Apparently, the necessity to delay the service time at the same pickup node at different iterations of Pass 2 does not imply a negative-weight cycle as it can result from different adjustments that are propagated through the constraint system. Consider again the example in Table 1: In the first iteration of Pass 2, τ_m needs to be

	Node	0	i	j	m	i^-	j^-	m^-	$2n+1$
	$[a.,b.]$	$[0,100]$	$[10,20]$	$[20,30]$	$[31,41]$	$[41,51]$	$[51,61]$	$[61,71]$	$[0,100]$
	$\underline{L}./\overline{L}.$	—	$0/30$	$31/100$	$0/30$	—	—	—	—
After pass 1	$\tau.$	0	*10*	20	(1) 31	*41*	51	61	71
1st pass 2, at node i	$\tau.$	0	(1) 11	21	*31*	41	(1) 52	*62*	72
2nd pass 2, at node m	$\tau.$	0	(1) *11*	21	(1) 32	*42*	52	62	72
2nd pass 2, at node i	$\tau.$	0	(2) 12	22	*32*	42	(1) 53	*63*	73

Table 2: Example showing nested requests leading to a cyclic shifting of waiting times in Algorithm 2

delayed by 1 unit of time because of a violated maximum RT of request m. In the second iteration of Pass 2, another delay of τ_m by 1 unit of time is necessary, this time caused by a violation of the maximum RT of request i that propagates through the constraint system in the two iterations. Even more, there is no direct way in Algorithm 2 to track the propagation of a shifting and therewith to decide if the repeated adjustment of the service time at the same node is caused by a negative-weight cycle or by the propagation of several different and independent shiftings.

However, if there is no negative-weight cycle, the number of iterations of Pass 2 that can occur is bounded, which can be seen as follows. After the first pass of Algorithm 2, we have a schedule of service times satisfying constraints (1), (2), and (4). For one or more requests, the schedule may be infeasible with respect to the maximum RT constraint (3). Algorithm 2 tries to repair this defect by delaying the service time at the corresponding pickup nodes in the first iteration of the second pass. In the further course of the algorithm, each of these time shiftings may be successively propagated to other nodes through the constraint system. Clearly, if there is no negative-weight cycle, none of the nodes can require more than one such time shifting for each of the infeasibilities detected in the first iteration. Even more, an additional iteration of the backward pass can only be caused by a time shifting at a pickup node. This can directly be seen in Algorithm 2 from the condition of the loop of the backward pass ($i^* > 0$, Line 6) and a possible update of i^* (Line 13) only for pickup nodes (Line 9). Thus, in this case the number of possible iterations is obviously bounded by the number of customer requests r in the route. As a result, the algorithm can stop after the $(r+1)$st iteration.

Overall, the adapted algorithm of Tang *et al.* (2010) without inner forward loop (*Algo2-w/o*) given in Algorithm 2 traverses the route once in forward direction with complexity $\mathcal{O}(r)$, followed by $\mathcal{O}(r)$ iterations of Pass 2 each requiring linear effort. Thus, we have the following result.

Proposition 1. *Algorithm 2 (*Algo2-w/o*) solves the scheduling problem (1)–(4) of the SPDP in* $\mathcal{O}(r^2)$ *time.* □

In the algorithmic variant *Algo2-w*, i.e., with the inner forward loop performed after each adjustment in the backward Pass 2, it is possible to further reduce the number of iterations needed to detect a negative-weight cycle in the constraint system. Recall that a subsequent iteration of the backward pass is only forced if in the current iteration the maximum RT constraint (3) of some request is violated and tried to be repaired by adjusting the respective service at the corresponding pickup. This adjustment may then propagate through the constraint system. To trigger an additional iteration in the subsequent one, the propagated adjustment must lead to a violation of the maximum RT of another request. Even more, the pickup of this request has to be later on the route (i.e., earlier in the backward pass). Otherwise, this RT-violation would be detected and tried to be repaired already in the further course of the current iteration of the backward Pass 2 so that no additional iteration is forced. Consequently, it takes at least two requests i and j to force an additional iteration of Pass 2 in the subsequent iteration: Request i whose pickup is delayed because its maximum RT constraint is violated and another request j that re-introduces the waiting time that was shifted before node v_i later on the route due to its minimum RT constraint. This re-introduction of waiting time might then lead to a violation of the maximum RT constraint of a third request which is picked up later than v_i so that the violated RT constraint is detected and repaired in the subsequent iteration of

Pass 2 and, hence, an additional iteration is forced. Summarizing, any adjustment that propagates through the constraint system and is not part of a negative-weight cycle can cause a maximum of $\lceil r/2 \rceil$ additional iterations of Pass 2, because (i) it can trigger at most one time shifting at each of the nodes and (ii) any additional iteration requires shifting the service time of customer nodes of at least two different requests. Thus, the algorithm *Algo2-w* can stop already after the $(\lceil r/2 \rceil + 1)$st iteration.

Overall, the worst-case number of necessary iterations of Pass 2 decreases to $\lceil r/2 \rceil + 1$ for *Algo2-w* (compared to the $2r + 1$ iterations needed for variant *Algo2-w/o*), but is still in $\mathcal{O}(r)$. However, the inner forward loop in Pass 2 leads to a weaker worst-case runtime of $\mathcal{O}(r^3)$ for *Algo2-w*.

3.3. Computational Comparison

To compare the practical performance of the approaches of Sections 3.1 and 3.2, we evaluate their runtime on a huge number of randomly generated routes with lengths reaching from $r = 15$ to $r = 200$ requests. Details on the generation of the routes and more detailed results can be found in the online supplement of this paper. Note that in our analysis we do not include routes that are identified as infeasible already in the first pass of the adapted Algorithm 2.

Table 3 summarizes our results. *BF* and *BFCT* denote our implementations of BF and BFCT that solve the feasibility problem by cycle-detection in the constraint graph associated with (1)–(4) reformulated as difference constraints. *Algo2-w* and *Algo2-w/o* denote the two versions of the adapted algorithm of Tang *et al.* (2010) as described in the previous section. Each row aggregates over a total of 20 000 routes. In order to obtain reliable computation times, we run each algorithm 10 000 consecutive times on each individual route. Table 3 reports the average time in milliseconds per instance needed by the algorithms for these 10 000 runs. In addition, we give the ratios for the average run times of algorithms *BFCT*, *Algo2-w*, and *Algo2-w/o* relative to *BF*. Finally, we report the average number of iterations of the second pass for *Algo2-w* and *Algo2-w/o*.

Table 3 reveals that for small to medium-sized values of $r < 50$, the additional effort in BFCT to aggressively search for cycles after each relabeling does not pay off and the straightforward implementation of BF is faster for these routes. With increasing r, however, BFCT is superior to BF. Regarding the two versions of Algorithm 2, it can be seen that *Algo2-w/o* performs slightly better than *Algo2-w* in terms of computation times except for very short routes although it requires more iterations of Pass 2 on average. Overall, the dedicated feasibility tests *Algo2-w/o* and *Algo2-w* that work directly on the given route are clearly superior to the approaches that rely on general shortest-path algorithms in the constraint graph.

	Avg. time in ms				Time rel. to *BF*			# Iterations	
r	*BF*	*BFCT*	*Algo2-w*	*Algo2-w/o*	*BFCT*	*Algo2-w*	*Algo2-w/o*	*Algo2-w*	*Algo2-w/o*
15	40.4	123.9	4.3	4.9	3.07	0.11	0.12	1.9	2.5
25	102.7	203.3	8.4	9.0	1.98	0.08	0.09	2.3	3.0
50	405.9	426.6	23.4	21.0	1.05	0.06	0.05	2.8	3.8
100	1694.1	867.3	83.8	66.4	0.51	0.05	0.04	4.6	6.2
200	7180.1	1754.5	195.0	165.1	0.24	0.03	0.02	6.2	7.7

Table 3: Aggregated computational results

4. Forward Time Slack

The concept of FTS was originally introduced by Savelsbergh (1992) in the context of the VRPTW. Let $\mathcal{T}_R = (\tau_1, ..., \tau_{2r+2})$ be a feasible schedule for route R. Savelsbergh (1992) defines the FTS $F_i^{\mathcal{T}}$ for a node

$v_i \in R$ as the maximum value by which the start of service τ_i at v_i can be increased without causing the route to become infeasible. Different authors have generalized the concept of FTS to other problems, e.g., Cordeau and Laporte (2003) for the DARP or Masson *et al.* (2013) for the PDPT.

VRPTW. The slack at a node $v_j, j \geq i$ with respect to node v_i is given by the cumulative waiting time between v_i and v_j and the difference between the end of the time window and the start of service at node v_j. The FTS $F_i^{\mathcal{T}}$ at node v_i is the minimum of all slacks at nodes v_j with $i \leq j \leq 2r+2$. Given a schedule $\mathcal{T}$, denote by $W_i^{\mathcal{T}}$ the waiting time at node v_i, by $tWT_{ij}^{\mathcal{T}} = \sum_{v_k \in R: i<k\leq j} W_k^{\mathcal{T}}$ the total waiting time between nodes v_i and v_j, and by $sTW_i^{\mathcal{T}} = b_{v_i} - \tau_i$ the time window slack at node v_i. Then (Savelsbergh, 1992):

$$F_i^{\mathcal{T}} = \min_{i \leq j \leq 2r+2} \left\{ tWT_{ij}^{\mathcal{T}} + sTW_j^{\mathcal{T}} \right\}. \tag{13}$$

Note that once the service time τ_i at v_i is fixed this definition of $F_i^{\mathcal{T}}$ is independent of the service times $\tau_j, j < i$, i.e., it is unique with respect to the route segment preceding v_i. This property is related to the fact that scheduling all service times as early as possible is an optimal strategy regarding the feasibility of routes for many VRPTW variants. It is also a crucial property for the standard constant-time feasibility test for the Pickup and Delivery Problem with Time Windows (PDPTW) and many of its variants when inserting a request i into a given feasible route R, i.e., inserting its corresponding pickup and delivery nodes i and i^- into route R at two given positions (Lu and Dessouky, 2006): In a preprocessing step, the earliest-possible schedule $\mathcal{T}_R^e$ of the original route R as well as the associated (unique) FTS-values are computed (which requires $\mathcal{O}(r^2)$ computation time). The actual feasibility test then proceeds by first inserting pickup node i as early as possible at the given position and evaluating the resulting time shift at the succeeding node by means of the precomputed FTS-values. The same procedure is then repeated with the delivery node i^-.

Note that it is not necessary to update the FTS-values after the pickup node i has been inserted in order to check the validity of the insertion of the delivery node i^-. The reason is that the FTS at the successor node j of i^-, i.e., the maximum feasible value by which the start of service at j can be delayed, does only depend on the schedule segment succeeding j and not the part that precedes it. Thus, the only relevant information is the actual value of the delay at j. It is not important if this delay is caused by the insertion of only one node, a pair of nodes, or even more nodes prior to j. As a result, the actual feasibility test for inserting one request at given positions runs in constant time provided that $\mathcal{T}_R^e$ and the associated FTS-values have been precomputed. Note further that the FTS-values are valid for testing the insertion of any request i at all possible positions for the pickup and delivery nodes i and i^-.

DARP. For problems with RTs, shifting the start of service at some node may cause infeasibilities also with respect to the RT constraints. This has to be incorporated in the definition of the FTS for such problems. For example, postponing the service at node v_i may increase the RT of a request j with $j < i$ and $j^- > i$. Consequently, Cordeau and Laporte (2003) define the following generalization of the FTS for the DARP:

$$F_i^{\mathcal{T}} = \min_{i \leq j \leq 2r+2} \left\{ tWT_{ij}^{\mathcal{T}} + \min\left(sTW_j^{\mathcal{T}}, sRT_{ij}^{\mathcal{T}} \right) \right\}, \tag{14}$$

where $sRT_{ij}^{\mathcal{T}}$ is the maximum RT slack given by $\overline{L}_{v_{j^+}} - \left(\tau_j - \tau_{j^+}\right)$ if $v_j \in D; i > j^+$ and $+\infty$ otherwise.

In contrast to (13), the FTS (14) of a node v_i for the DARP (and similarly for other problems with maximum RTs) is not unique with respect to the route segment preceding v_i. Indeed, it may be possible for some $v_j \in D$ with $j^+ < i$ and $j > i$ to increase τ_{j^+} such that τ_i stays the same while $sRT_{ij}^{\mathcal{T}}$ and $F_i^{\mathcal{T}}$ also increase. For that reason, the standard feasibility test of request insertions for PDPTW variants based on FTS (see Lu and Dessouky, 2006, and sketched above) is not exact in the presence of maximum RT constraints.

Consider the route given in Table 4. Feasibility of the insertion of another request k is to be evaluated. Assume a travel time of ten between all nodes, $\overline{L}_i = 40$, and time windows $[0, 20]$ and $[0, 100]$ at the nodes k and k^-, respectively. Inserting node k before i and node k^- before j^- results in a feasible route. When using the FTS-values of the earliest-possible schedule $\mathcal{T}^e$ (with $\tau_i^e = 10$), however, the insertion appears to

	Node	0	i	j	j^-	i^-	$2n+1$
	$[a.,b.]$	$[0,100]$	$[10,30]$	$[30,50]$	$[40,60]$	$[50,70]$	$[0,100]$
	$\underline{L}_\cdot/\overline{L}_\cdot$	—	$0/40$	$0/100$	—	—	—
(Earliest-possible) schedule $\mathcal{T}^e$	$\tau^e_\cdot$	0	10	(10) 30	40	50	60
FTS for nodes after v_i with $\tau^e_i = 10$	$F^{\mathcal{T}^e}_\cdot$	—	—	0	0	0	40
Schedule $\mathcal{T}'$	$\tau'_\cdot$	0	20	30	40	50	60
FTS for nodes after v_i with $\tau'_i = 20$	$F^{\mathcal{T}'}_\cdot$	—	—	10	10	10	40

Table 4: Example route for which feasibility checking of request insertions based on FTS is problematic

be infeasible because $\tau^e_{j^-}$ is increased by $10 > F^{\mathcal{T}^e}_{j^-} = 0$. The reason is that the restriction to the earliest possible service time $\tau^e_i = 10$ at node i results in an underestimation of the FTS at node j^- that is valid for this particular insertion. As described above, larger values for the service times at the nodes preceding j^- can lead to larger FTS-values for the same service time at node j^-. Indeed, considering FTS-values based on schedule $\mathcal{T}'$ that services node i at time 20 correctly determines feasibility of this insertion. On the other hand, using the latter FTS-values results in misjudging the insertion of k before j and k^- before j^- as feasible. In this case, the problem is that $F^{\mathcal{T}'}_{j^-} = 10$ relies on the service time $\tau'_i = 20$ at i which is not possible with the additional visit of node k before j in the new route. Generally speaking, the insertion of an additional node k before a node j limits the service times of all nodes preceding j on the route and, therewith, restricts the set of FTS-values for j and all succeeding nodes that can be legitimately used for evaluating the particular insertion.

Summing up, the main issue for feasibility testing in the presence of maximum RTs is the following: Choosing different schedules of the original route can lead to different FTS-values for a specific node, even with identical service time at that node. Depending on the actual insertion that is tested, however, not all of them are valid. Clearly, this effect is not known during the preprocessing phase in which FTS-values should be computed. Moreover, it is different for each request i to be inserted and all the possible positions for the insertion of the corresponding pickup and delivery nodes i and i^-. As a result, no single FTS-value that could be used for the exact feasibility evaluation of all possible insertions exists for the nodes of the original route. The consequence is that the adaptation of the standard FTS-based procedure for PDPTW variants does not lead to a valid feasibility check of request insertions in the presence of maximum RT constraints. However, the same technique can be used as a constant-time verifiable sufficient condition: Given a route R, compute the earliest-possible schedule $\mathcal{T}^e$. The FTS-value $F^{\mathcal{T}^e}_i$ of a node $v_i \in R$ given the earliest-possible schedule is clearly valid for all insertions of a node prior to v_i on the route. Thus, no insertion can be incorrectly classified as feasible when using $F^{\mathcal{T}^e}_i$ (unlike in the second part of the above example). The value $F^{\mathcal{T}^e}_i$ might, however, be a too pessimistic estimation of the actual slack time (see first part of the above example). The insertion of new requests can then be evaluated based on the earliest-possible schedule and its associated FTS-values. Additionally, the maximum RT constraint of the newly inserted request has to be checked. As shown in the example above, failing these tests does not imply infeasibility of the insertion.

SPDP. The presence of minimum RTs raises two additional issues compared to the DARP. First, shifting the service time at a pickup node is constrained by the minimum RT of this request. Second, not necessarily all of the cumulative waiting time between two nodes v_i and v_j is slack. Decisive is that some waiting time may have to be included between them because of the minimum RT of some requests. Denote by $UW^{\mathcal{T}}_{ij}$ the *usable waiting time* between v_i and v_j, i.e., the waiting time between v_i and v_j that can be incorporated into the slack. Then, the FTS for a node v_i is given by

$$F^{\mathcal{T}}_i = \min_{i \le j \le 2r+2} \left\{ UW^{\mathcal{T}}_{ij} + \min\left(sTW^{\mathcal{T}}_j, sRT^{\mathcal{T}}_{ij}\right) \right\}, \tag{15}$$

with

$$sRT_{ij}^{\mathcal{T}} = \begin{cases} b_{v_{j^-}} - \left(\underline{L}_{v_j} + \tau_j\right) & \text{if } v_j \in P, \\ \overline{L}_{v_{j^+}} - \left(\tau_j - \tau_{j^+}\right) & \text{if } v_j \in D; i > j^+, \\ +\infty & \text{otherwise.} \end{cases}$$

To determine $UW_{ij}^{\mathcal{T}}$, we first define for each delivery node v_{j^-} the *necessary waiting time*

$$NW_{j^-} = \left(\underline{L}_j - \sum_{v_k \in R: j < k \leq j^-} t_{k-1,k} \right)^+$$

for request j which is the amount of waiting time that inevitably occurs between pickup and delivery of request j and, hence, can never be part of the slack between nodes v_i and v_{j^-} if $j^- > i$. Note that NW_{j^-} is independent of the actual schedule $\mathcal{T}_R$ for which the FTS is determined. If there is another request k with positive NW_{k^-} in between v_i and v_{j^-}, it has to be ensured that the overlapping necessary waiting times NW_{j^-} and NW_{k^-} are not considered twice in the definition of $UW_{ij}^{\mathcal{T}}$. Therefore, we define for each delivery node v_{j^-} after v_i the already *included waiting time* between them, which is also independent of the base schedule $\mathcal{T}_R$, as

$$IW_{ij^-} = \sum_{\substack{v_{k^-} \in R: i < k^- < j^- \\ v_{k^-} \in D, k^+ \geq i, k^- > j^+}} \left(NW_{k^-} - IW_{ik^-}\right)^+ .$$

The term $\left(NW_{j^-} - IW_{ij^-}\right)^+$ then gives the amount of waiting time that will always be present at a delivery node $v_{j^-} \in D$ (because of the minimum RTs of the requests) if there is no other waiting time in between nodes v_{j^+} and v_{j^-} on a schedule. Consequently, this needed amount of waiting time is never slack and has to be deducted from the cumulative waiting time of any considered schedule when computing usable waiting times.

In addition to these schedule-independent times, there may also be non-slack waiting times that are induced by the considered base schedule $\mathcal{T}_R$. This can happen if in the considered schedule $\mathcal{T}_R$, there is a deficit in waiting times for a request j and a node v_k, $k \geq i$ in between pickup and delivery of j, i.e., $v_j \leq v_k < v_{j^-}$. It means that the difference

$$\Delta_{ijk}^{\mathcal{T}} = tWT_{jj^-}^{\mathcal{T}} - \sum_{\substack{v_{m^-} \in R: k < m^- \leq j^- \\ v_{m^-} \in D, m^+ \geq i}} \left(NW_{m^-} - IW_{im^-}\right)^+$$

is negative. The value $\Delta_{ijk}^{\mathcal{T}}$ is the difference of the cumulative waiting time between v_j and v_{j^-} and the cumulative needed amount of waiting time at the delivery nodes $v_{m^-} \in D$ with $m^+ > i$ on the partial route between node v_k and the delivery node v_j^- of request j. This is possible whenever request j is nested with another request p whose delivery node v_{p^-} lies in between v_k and v_{j^-} on route R and has a positive need for waiting time $NW_{p^-} - IW_{ip^-} > 0$.

If for $\mathcal{T}_R$ the deficit in waiting times $\left(\Delta_{ijk}^{\mathcal{T}}\right)^-$ is larger than $\delta_j = \overline{L}_j - (\tau_{j^-} - \tau_j)$, i.e., the slack with respect to the maximum RT constraint of v_j, the amount $\left(\Delta_{ijk}^{\mathcal{T}}\right)^- - \delta_j$ must not be incorporated into the usable waiting time of node v_k. The reason is that shifting this waiting time (by delaying the service at v_i) before nodes v_j, v_k, and v_p causes the re-introduction of waiting times right before the delivery node v_{p^-} to meet the associated minimum RT $\underline{L}_{v_p}$ of request p due to $NW_{p^-} - IW_{ip^-} > 0$. This in turn increases the RT of request j compared to the initial schedule $\mathcal{T}_R$ so that the services at all nodes on the partial route

from v_j to v_k have to be delayed by $\left(\Delta^{\mathcal{T}}_{ijk}\right)^- - \delta_j$.

Formally, we have for each $j > i$ the *schedule-dependent additional amount of time*

$$AW^{\mathcal{T}}_{ij} = \left(\max_{\substack{v_k \in R: i<k\leq j \\ v_k \in P, k^- > j}} \left(\Delta^{\mathcal{T}}_{ikj}\right)^- - \delta_k \right)^+$$

that has to be deducted from the cumulative waiting times at node v_j. For convenience, we define $AW^{\mathcal{T}}_{ij} = 0$ for all $j \leq i$. Furthermore, if for a request j the times $AW^{\mathcal{T}}_{ij}$ and $AW^{\mathcal{T}}_{ij^-}$ differ they may imply additional, schedule-dependent waiting times that are necessary between the pickup and delivery nodes. Consequently, *schedule-dependent necessary waiting times* and *schedule-dependent already included waiting times* at delivery nodes $v_{j^-}, j^- > i$ are defined as

$$NW^{\mathcal{T}}_{ij^-} = NW_{j^-} + \left(AW^{\mathcal{T}}_{ij^+} - AW^{\mathcal{T}}_{ij^-}\right)^+$$

and

$$IW^{\mathcal{T}}_{ij^-} = \sum_{\substack{v_{k^-} \in R: i<k^-<j^- \\ v_{k^-} \in D, k^+ \geq i, k^- > j^+}} \left(NW^{\mathcal{T}}_{ik^-} - IW^{\mathcal{T}}_{ik^-}\right)^+,$$

respectively.

Then, the *usable waiting time* $UW^{\mathcal{T}}_{ij}, j \geq i$ is given by

$$UW^{\mathcal{T}}_{ij} = tWT^{\mathcal{T}}_{ij} - \sum_{\substack{v_{k^-} \in R: i<k^-\leq j \\ v_{k^-} \in D, k^+ \geq i}} \left(NW^{\mathcal{T}}_{ik^-} - IW^{\mathcal{T}}_{ik^-}\right)^+ - AW^{\mathcal{T}}_{ij}.$$

Consider the small example in Table 5 that demonstrates the computation of the usable waiting times $UW^{\mathcal{T}}_{0\cdot}$ between the origin depot and the other nodes based on the earliest-possible schedule given by the service times $\tau^e_\cdot$. The travel times between all nodes are assumed to be 10. All remaining data is specified in the table. The interesting part of the computation relates to the waiting time that is not slack due to the interplay of the minimum RT of request i and the maximum RT of request j. From the node order in the route, it follows that there must be a waiting time of $NW_i = 13$ between pickup and delivery of request i so that its minimum RT is satisfied. In the earliest-possible schedule, 3 units of this necessary waiting time occur between nodes i and j while the remaining 10 units lie between nodes j and i^-. Moreover, also the cumulative waiting time between j and j^- is only 10 so that the consumption of the 3 units of waiting time between nodes i and j by delaying the service at the origin depot would increase the actual RT of request j by this 3 time units because of the re-introduction of the same amount of waiting time before node i^- to meet the minimum RT $\underline{L}_i$. Consequently, there is a deficit in waiting times of this 3 time units, i.e., $\Delta^{\mathcal{T}}_{0j\cdot} = 3$, for nodes j, k, and m in between pickup and delivery of request j. As the slack regarding the maximum RT constraint of j is only $\delta_j = 2$, one has to deduct an additional $AW^{\mathcal{T}}_{0\cdot} = \Delta^{\mathcal{T}}_{0j\cdot} - 2 = 1$ unit of waiting time at these nodes to obtain $UW^{\mathcal{T}}_{0\cdot}$. The intuition behind the different times used in the computation of the usable waiting time can be seen quite nicely also from the actual service times $\tau^5_\cdot$ that result from delaying the service at the origin depot by 5 units of time which equals the cumulative waiting time between the origin depot and node j, but is 1 unit more than the usable waiting time $UW^{\mathcal{T}}_{0j} = 4$: With $\tau^5_0 = 5$, node j could still be reached at time 25 from its predecessors. However, since $\tau^5_i = 15$ and $\underline{L}_i = 63$, service at i^- cannot start before $\tau^5_{i^-} = 78$ meaning that the waiting time that was removed between i and j has to be re-inserted somewhere before i^-. Consequently, service at j^- cannot start before $\tau^5_{j^-} = 88$ which implies that the ride time of j would increase by $\Delta^{\mathcal{T}}_{0jj} = 3$ units of time compared to the earliest-possible schedule, while only an

Node	0	i	j	k	m	m^-	i^-	j^-	k^-	$2n{+}1$
$[a_\cdot, b_\cdot]$	$[0, 200]$	$[12, 24]$	$[25, 35]$	$[35, 45]$	$[45, 55]$	$[55, 70]$	$[75, 87]$	$[85, 97]$	$[95, 110]$	$[0, 200]$
$\underline{L}_\cdot/\overline{L}_\cdot$	—	63/200	0/62	65/200	15/200	—	—	—	—	—
$\tau^e_\cdot$	0	(2) 12	(3) 25	35	45	(5) 60	(5) 75	85	(5) 100	110
$NW_\cdot$	—	—	—	—	—	5	13	0	15	—
$IW_{0\cdot}$	—	—	—	—	—	0	5	13	13	—
$\Delta^{\mathcal{T}}_{0j\cdot}$	—	—	-3	-3	-3	0	0	—	—	—
$AW^{\mathcal{T}}_{0\cdot}$	0	0	1	1	1	0	0	0	0	0
$NW^{\mathcal{T}}_{0\cdot}$	—	—	—	—	—	6	13	1	16	—
$IW^{\mathcal{T}}_{0\cdot}$	—	—	—	—	—	0	6	13	13	—
$UW^{\mathcal{T}}_{0\cdot}$	0	2	4	4	4	4	2	2	4	4
$\tau^4_\cdot$	(4) 4	14	(1) 25	35	45	(5) 60	(7) 77	87	(3) 100	110
$\tau^5_\cdot$	(5) 5	15	(1) 26	36	46	(5) 61	(7) 78	88	(3) 101	111

Table 5: Example demonstrating the computation of $UW^{\mathcal{T}}_{0\cdot}$ in the case of nested requests. The travel times between all nodes are assumed to be 10.

increase of $\delta_j = 2$ units is feasible. Thus, the start of service τ^5_j at node j has to be delayed by $AW^{\mathcal{T}}_{0j} = 1$ unit of time compared to the earliest time it could be reached from its predecessor i which is exactly the difference between the delay at the origin and the usable waiting time between $UW^{\mathcal{T}}_{0j}$. In contrast, when delaying the service at the origin depot by no more that $UW^{\mathcal{T}}_{0j}$, the service time at j remains unchanged as can be seen from the service times $\tau^4_\cdot$.

As in the DARP, the FTS (15) for a node v_i is not unique with respect to the route segment preceding v_i and the standard technique for PDPTW variants cannot be used for exact feasibility testing of request insertions. However, it enables a constant time verifiable sufficient condition similar to the one described above for the DARP that can be used within insertion-based approaches to the SPDP in order to reduce the number of calls to one of the computationally expensive feasibility tests of Section 3. To analyze the possible benefits of using such a sufficient condition, we test it on the route set of Section 3.3 as follows. Given a feasible route, we remove one of the requests, calculate the earliest-possible schedule and associated FTS-values for the resulting route and evaluate the reinsertion of the removed nodes at their original positions based on the sufficient condition. This is done for all routes and all requests on the routes (one at a time). Overall, the condition appears to be very effective as it was able to prove feasibility of the insertions in approximately 96% of the 3 900 000 calls. More detailed results are presented in the online supplement of this paper. They reveal that the condition is especially effective for routes with few nested requests.

5. Conclusions

The SPDP is the prototypical VRP variant with temporal intra-route synchronization constraints. Because of these constraints, the efficient feasibility testing of routes in the SPDP is non-trivial. Many exact and heuristic solution approaches to VRP variants, however, rely on such efficient feasibility tests. In this paper, we derived different route feasibility checks for the SPDP and compared their practical performance on a huge number of randomly generated routes. The described feasibility tests can provide a basis to adapt existing solution frameworks for VRP variants to tackle the SPDP.

Moreover, we extended to the SPDP the concept of FTS that has proven to be a useful tool for efficient feasibility testing in approaches that are based on insertion heuristics like, e.g., (adaptive) large neighborhood search which is one of the most widely used heuristics for VRP variants. We demonstrated that in problems with maximum RTs the FTS of a node v_i is not unique with respect to the route segment preceding v_i. As a consequence, the standard feasibility test of request insertions for PDPTW variants based on FTS is not exact in these problems. However, the same technique can be used as a constant-time verifiable sufficient

condition. Complementing this strategy with other sufficient or necessary conditions might significantly reduce the number of calls to a computationally more expensive exact feasibility test in an insertion-based approach. Similar strategies for problems with costly exact feasibility test have been used, e.g., by Braekers *et al.* (2014) for the DARP or Masson *et al.* (2014) for a DARP with transfer possibilities.

Because of the prototypical character of the SPDP, we believe that the presented concepts are also relevant for other routing problems with temporal synchronization constraints. For example, the feasibility tests of Section 3.1 can be adapted to a SPDP with transfer possibilities between routes in a straightforward way. Likewise, the proposed definition of the FTS can be generalized to the problem with transfer possibilities using the technique of Masson *et al.* (2013).

Acknowledgement

This research was partially funded by the Deutsche Forschungsgemeinschaft (DFG) under grant no. IR 122/5-2.

References

Barnhart, C., Cohn, A. M., Johnson, E. L., Klabjan, D., Nemhauser, G. L., and Vance, P. H. (2003). Airline crew scheduling. In R. W. Hall, editor, *Handbook of Transportation Science*, pages 517–560. Kluwer Academic Publishers, Norwell, MA.

Bélanger, N., Desaulniers, G., Soumis, F., and Desrosiers, J. (2006). Periodic airline fleet assignment with time windows, spacing constraints, and time dependent revenues. *European Journal of Operational Research*, **175**(3), 1754–1766.

Braekers, K., Caris, A., and Janssens, G. K. (2014). Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transportation Research Part B*, **67**, 166–186.

Bredström, D. and Rönnqvist, M. (2008). Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, **191**, 19–29.

Cherkassky, B. V., Georgiadis, L., Goldberg, A. V., Tarjan, R. E., and Werneck, R. F. (2009). Shortest-path feasibility algorithms. *Journal of Experimental Algorithmics*, **14**, 2.7–2.37.

Cordeau, J.-F. and Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B*, **37**(6), 579–594.

Cordeau, J.-F. and Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, **153**(1), 29–46.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2001). *Introduction To Algorithms*. MIT Press.

Eveborn, P., Flisberg, P., and Rönnqvist, M. (2006). Laps care – an operational system for staff planning of home care. *European Journal of Operational Research*, **171**(3), 962 – 976.

Firat, M. and Woeginger, G. J. (2011). Analysis of the dial-a-ride problem of Hunsaker and Savelsbergh. *Operations Research Letters*, **39**(1), 32–35.

Gschwind, T. (2015). A comparison of column-generation approaches to the synchronized pickup and delivery problem. *European Journal of Operational Research*, **247**(1), 60–71.

Gschwind, T. and Irnich, S. (2015). Effective handling of dynamic time windows and its application to solving the dial-a-ride problem. *Transportation Science*, **49**(2), 335–354.

Kovacs, A. A., Golden, B. L., Hartl, R. F., and Parragh, S. N. (2014). Vehicle routing problems in which consistency considerations are important: A survey. *Networks*, **64**(3), 192–213.

Lu, Q. and Dessouky, M. M. (2006). A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, **175**(2), 672–687.

Masson, R., Lehuédé, F., and Péton, O. (2013). Efficient feasibility testing for request insertion in the pickup and delivery problem with transfers. *Operations Research Letters*, **41**(3), 211–215.

Masson, R., Lehuédé, F., and Péton, O. (2014). The dial-a-ride problem with transfers. *Computers & Operations Research*, **41**, 12–23.

Pillac, V., Guéret, C., and Medaglia, A. L. (2012). A parallel matheuristic for the technician routing and scheduling problem. *Optimization Letters*, **7**(7), 1525–1535.

Rasmussen, M. S., Justesen, T., Dohn, A., and Larsen, J. (2012). The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. *European Journal of Operational Research*, **219**(3), 598–610.

Savelsbergh, M. W. P. (1992). The vehicle routing problem with time windows: Minimizing route duration. *INFORMS Journal on Computing*, **4**(2), 146–154.

Tang, J., Kong, Y., Lau, H., and Ip, A. W. H. (2010). A note on "efficient feasibility testing for dial-a-ride problems". *Operations Research Letters*, **38**(5), 405–407.

Tarjan, R. E. (1981). Shortest paths. Technical report, AT&T Bell Laboratories, Murray Hill, NJ.

Online Supplement

A. Detailed Computational Results

In this section, we give additional details on the computational comparison of the different feasibility checks of Sections 3.1 and 3.2 and on the performance of the FTS-based sufficient condition for evaluating the feasibility of request insertions as described in Section 4. The algorithms are tested on a huge number of randomly generated routes. Furthermore, a separate analysis is conducted for integer and double precision input parameters. Euclidean distances (rounded up in the case of integer inputs) are used as travel times between the nodes. Minimum and maximum RTs are specified proportional to the Euclidean distances between pickup and delivery location. Details on the parameters used for the generation of the routes are given in Table 6. The columns have the following meaning:

r	Number of requests
Cust. loc.	Intervals from which the customer locations are drawn. The depots are located at $(0, 0)$.
Horizon	Time horizon
TW	Mean value for the length of the time windows
Min RT	Mean value for the factor specifying the minimum RT
Max RT	Mean value for the factor specifying the maximum RT

n	Cust. loc.	Horizon	TW	Min RT	Max RT
15	$[10, 10] \times [10, 10]$	450	60	2	10
25	$[10, 10] \times [10, 10]$	750	100	2	10
50	$[10, 10] \times [10, 10]$	1500	200	2	10
100	$[10, 10] \times [10, 10]$	3000	400	2	15
200	$[10, 10] \times [10, 10]$	6000	500	2	30

Table 6: Details on the parameters used for route generation

For each value of r, we generate routes with different characteristics regarding the number of requests that are open at the nodes in the following way: At each node another request is picked up with probability p, otherwise one of the open requests is delivered. The considered values of p reach from 0.15 to 0.50 in steps of 0.05. For each probability p, a pretest filters 625 feasible and 625 infeasible routes so that a total of 10 000 routes is considered for each value of r. Note that we consider only routes that are not identified as infeasible already in the first pass of the adapted Algorithm 2.

Tables 7 and 8 summarizes our results on the comparison of the different feasibility checks of Sections 3.1 and 3.2. The columns and rows have the following meaning:

r	Number of requests
Max # open	The maximum number of open requests at a node; we give average, maximum, and minimum values over the routes
BF (BFCT)	Our implementation of BF (BFCT) that solves the feasibility problem by cycle-detection in the constraint graph (see Section 3.1)
Algo2-w (Algo2-w/o)	The version of Algorithm 2 with (without) the inner forward loop performed after each adjustment in Pass 2 (see Section 3.2)
Solution time	The time in milliseconds to run the respective algorithm 10 000 consecutive times on a route; we present average, maximum, and minimum values over the routes
# Iterations	The number of iterations of the second pass for *Algo2-w* and *Algo2-w/o*

			$r =$					
			15	25	50	100	200	avg.
	Max # open	avg.	2,7	3,3	4,2	5,5	6,8	4,5
		max	6,0	8,0	11,0	19,0	23,0	13,4
		min	1,0	1,0	2,0	2,0	2,0	1,6
Solution time	*BF*	avg.	38,7	97,9	382,8	1569,4	6593,4	1736,4
		max	141,0	235,0	968,0	3953,0	19171,0	4893,6
		min	0,0	15,0	47,0	156,0	497,0	143,0
	BFCT	avg.	132,8	202,2	426,1	841,2	1755,3	671,5
		max	234,0	360,0	672,0	1172,0	2188,0	925,2
		min	109,0	172,0	374,0	687,0	1406,0	549,6
	Algo2-w	avg.	4,5	9,0	25,6	91,2	208,4	67,7
		max	47,0	110,0	625,0	2172,0	6449,0	1880,6
		min	0,0	0,0	0,0	15,0	31,0	9,2
	Algo2-w/o	avg.	5,0	9,5	22,6	73,2	175,4	57,1
		max	79,0	188,0	718,0	2828,0	11562,0	3075,0
		min	0,0	0,0	0,0	15,0	31,0	9,2
# Iterations	*Algo2-w*	avg.	1,8	2,3	3,0	4,9	6,6	3,7
		max	9,0	14,0	26,0	51,0	101,0	40,2
		min	1,0	1,0	1,0	1,0	1,0	1,0
	Algo2-w/o	avg.	2,1	2,7	3,6	6,2	7,7	4,5
		max	31,0	51,0	101,0	201,0	401,0	157,0
		min	1,0	1,0	1,0	1,0	1,0	1,0

Table 7: Detailed computational results for the different feasibility tests and integer inputs

			$r =$					
			15	25	50	100	200	avg.
	Max # open	avg.	2,7	3,3	4,2	5,5	6,8	4,5
		max	6,0	6,0	11,0	19,0	23,0	13,0
		min	1,0	1,0	1,0	2,0	2,0	1,4
Solution time	*BF*	avg.	42,2	107,5	428,9	1818,7	7766,8	2032,8
		max	110,0	328,0	1265,0	5235,0	24340,0	6255,6
		min	0,0	15,0	62,0	187,0	625,0	177,8
	BFCT	avg.	115,0	204,4	427,1	893,4	1753,7	678,7
		max	219,0	344,0	656,0	1156,0	2171,0	909,2
		min	93,0	172,0	359,0	734,0	1375,0	546,6
	Algo2-w	avg.	4,1	7,8	21,2	76,4	181,7	58,2
		max	47,0	78,0	453,0	2531,0	6037,0	1829,2
		min	0,0	0,0	0,0	0,0	15,0	3,0
	Algo2-w/o	avg.	4,9	8,6	19,3	59,6	154,8	49,4
		max	63,0	172,0	625,0	2578,0	12359,0	3159,4
		min	0,0	0,0	0,0	0,0	15,0	3,0
# Iterations	*Algo2-w*	avg.	1,9	2,3	2,7	4,3	5,8	3,4
		max	9,0	14,0	26,0	51,0	101,0	40,2
		min	1,0	1,0	1,0	1,0	1,0	1,0
	Algo2-w/o	avg.	2,8	3,4	4,0	6,1	7,6	4,8
		max	31,0	51,0	101,0	201,0	401,0	157,0
		min	1,0	1,0	1,0	1,0	1,0	1,0

Table 8: Detailed computational results for the different feasibility tests and double inputs

Table 9 presents details on the performance of the FTS-based sufficient condition proposed in Section 4 for evaluating the feasibility of request insertions into given feasible routes. The columns and rows have the following meaning:

Param.	Type of input data (integer or double precision) of the routes
p	Probability for delivering one of the open requests that was used in the generation of the routes. A higher value implies that on average more requests are open at the same time, i.e., the requests are more nested.
Max # open	The maximum number of open requests at a node; we give average, maximum, and minimum values over the routes
Insertion tests	The overall number of evaluated request insertions (total); the absolute (# pos.) and relative (% pos.) number of evaluations for which the sufficient condition successfully detects feasibility of the insertion

		Max # open			Insertion tests		
Param.	*p*	avg.	max	min	total	# pos.	% pos.
integer	0.15	2.8	7	1	243750	242540	99.50
	0.20	3.1	8	1	243750	241784	99.19
	0.25	3.4	10	1	243750	240771	98.78
	0.30	3.8	10	1	243750	239016	98.06
	0.35	4.1	11	1	243750	236936	97.20
	0.40	4.5	15	1	243750	234274	96.11
	0.45	4.8	13	2	243750	230505	94.57
	0.50	5.0	16	2	243750	226653	92.99
double	0.15	2.8	7	1	243750	242192	99.36
	0.20	3.2	8	1	243750	241354	99.02
	0.25	3.5	9	1	243750	239739	98.35
	0.30	3.8	12	1	243750	237747	97.54
	0.35	4.2	11	2	243750	234592	96.24
	0.40	4.5	13	1	243750	230732	94.66
	0.45	4.8	14	2	243750	226077	92.75
	0.50	5.1	13	2	243750	220482	90.45

Table 9: Detailed computational results for FTS-based sufficient condition